1. 上课约定须知
2. 上次作业复盘
3. 上次内容总结
4. 本次内容大纲
5. HBase-2.x 的读写流程源码分析
    5. 1. HBase 核心工作流程分析
        5. 1. 1. createTable 创建表流程分析
        5. 1. 2. put 写数据流程分析
        5. 1. 3. HBase Region 定位
        5. 1. 4. get / scan 读数据流程分析
    5. 2. HBase 内部核心机制详解
        5. 2. 1. flush 流程分析
        5. 2. 2. split 流程分析
        5. 2. 3. compact 流程分析
6. 本次课程总结
7. 本次课程作业

# 1. 上课约定须知

课程主题：HBase分布式NoSQL数据库--第三次课（源码解析）
上课时间：20:00 - 23:00
课件休息：21:30 左右 休息10分钟
课前签到：如果能听见音乐，能看到画面，请在直播间扣 666 签到

# 2. 上次作业复盘

当时学完 Zookeeper 的时候，给大家布置了作业，让各位实现一个类似于 ZooKeeper 的数据模型系统，具备一些基本的功能，比如：

```
1、通过树形方式来组织数据，具备基本的节点增删改查的功能
2、该系统具备冷启动恢复数据状态的功能。
```

现在来实现一个 HBase 存储系统！两点要求：

```
1、设计一个系统存储key_value类型数据，该系统为四维表模型，让系统具备根据一个条件，两个条件，三个条件，四个条件查找value的能力，如果不是根据四个条件来查找，返回单个value，如果不是根据四个条件来查询，则返回value的集合，最好不要只是value的集合，最好是一个map。
2、该系统可以冷启动恢复数据状态
```

未来进行改造：分布式的系统！

```
现在的作业 + RPC框架 + 四维表的最外层map实现成一种分布式的结构
```

# 3. 上次内容总结

上次课程是 HBase 的第二次课程，主要开始讲解 HBase-2.x 版本的源码讲解，主要知识点是：HBase 集群启动

```
1、HBase 集群的 集群启动脚本分析
2、HBase 集群的 HMaster 启动源码分析
    RPC 服务端启动起来
    启动一个服务等待 RegionServer 上线注册
    启动 WEB UI 系统
    竞选 active master
    ...
    各种其他的服务组价的初始化等
3、HBase 集群的 HRegioinServer 启动源码分析
    RPC 服务端启动起来
    启动 WebUI
    先进行注册
    然后进行心跳汇报
    在进行各种服务组件的初始化
    ....
```

# 4. 本次内容大纲

今天的主要内容是 HBase-2.x 版本源码分析的第二次课，是 HBase 课程的第三次课，主要讲解 HBase 的读写流程源码分析。主要内容点包括：

```
1、HBase 创建表流程分析
2、put 写数据流程分析
    region 定位
3、get 读数据流程分析
4、flush 流程
5、split 流程
6、compact 流程
```

```
1、DDL 语句的 源码分析
    createTable
2、DML 语句的 源码分析
    put（flush  split comact）/get
```

```
1、创建表
    createTable
2、读写流程
    put get 都会涉及到一个重要的面试点：region的定位  region的寻址机制
3、hbase的内部工作机制
    flush  split comact
```

# 5. HBase-2.x 的读写流程源码分析

## 5.1. HBase 核心工作流程分析

### 5.1.1. createTable 创建表流程分析

先获取链接对象：

```
关于创建表，必须先获取链接，链接的实现类是： ConnectionImplementation
最终会通过反射的方式来创建链接对象，构造方法中会做两件字重要的事情：
1、客户端可能会要跟zk打交道：regisitry 的对象 ： ZKAsyncRegistry
2、客户端肯定要跟服务端打交道：初始化一个 RpcClient
```

入口：HBaseAdmin.createTable()

详细流程：

```
1、createTableAsync(desc, splitKeys)
2、executeCallable(MasterCallable)
    caller.callWithRetries(callable, operationTimeout);
        callable.prepare(tries != 0);
            this.connection.getMaster();
        interceptor.intercept(context.prepare(callable, tries));
        callable.call(getTimeout(callTimeout));
            MasterCallable.rpcCall()
                master.createTable(getRpcController(), request);
3、跳转到：MasterRpcServices.createTable()
    master.createTable(tableDescriptor, splitKeys,...)
    MasterProcedureUtil.submitProcedure(new
MasterProcedureUtil.NonceProcedureRunnable())
    submitProcedure(new CreateTableProcedure())
    StateMachineProcedure.execute()
        CreateTableProcedure.executeFromState()
```

这里有一个重要的知识点：StateMachineProcedure

状态机的启动流程：

```
HMaster的构造方法
HMaster的start()
    startActiveMasterManager()
        finishActiveMasterInitialization(status);

            # 第一步：ProcedureExecutor 的初始化
            createProcedureExecutor();
                procedureExecutor = new ProcedureExecutor()
                procedureExecutor.init(numThreads, abortOnCorruption);
                    # 初始化 至少 16 个线程
                    workerThreads.add(new WorkerThread(threadGroup));
                    WorkerThread.start()
                        WorkerThread.run()
                            scheduler.poll()    # 从Procedure 队列 获取 Proceudre
                            executeProcedure(proc);
```

```
                                execProcedure(procStack, proc);
                                    procedure.doExecute(getEnvironment());
                                        Procdure.execute(env);
                                            StateMachineProcedure.execute()
                                                executeFromState(env, state);

CreateTableProcedure.executeFromState

            # 第二步： 启动ProcedureExecutor
            startServiceThreads();
                startProcedureExecutor();
                    procedureExecutor.startWorkers();
                        WorkerThread.start()
```

关于 CreateTableProcedure 的状态流转是这样的：

```
1、CREATE_TABLE_PRE_OPERATION          做校验，做校验（检验表是否存在，是否至少一个列簇）协处理器
2、CREATE_TABLE_WRITE_FS_LAYOUT        写 region 信息到 HDFS， 同时创建 Region 信息
3、CREATE_TABLE_ADD_TO_META            写表的 region 信息到 meta 表
4、CREATE_TABLE_ASSIGN_REGIONS         分派 Region 给 RegioinServer，默认随机分配
5、CREATE_TABLE_UPDATE_DESC_CACHE      把 表的 TableDescriptor 信息更新到 缓存
6、CREATE_TABLE_POST_OPERATION         协处理器 收尾
7、NO_MORE_STATE                       没有更多状态
```

创建表的核心动作应该有哪些?

```
1、做校验  这个表名的表不存在，表的列簇必须是合法的
2、假设你之前在删除该表名的表的时候，并没有在HDFS里面，把该表对应的一个工作目录删除干净。
    删除干净之前，创建一些对应的需要的空的目录和初始化一些文件
3、HMaster需要访问Meta表，把该表的元数据写进去
    创建表的时候，是可以指定是否有 splitKeys，如果有就有多个region，每一个region就会抽象成一个RegioinInfo（region的元数据）对象。插入到meta表
4、HMaster要分派这些region给哪些regionserver去管理
5、把当前这个表的一个meta信息，更新到缓存中
6、做一些收尾的动作
```

创建表涉及到3个重要的知识：

```
1、ProcedureExecutor的初始化： HMaster启动的时候，会启动 ProcedureExecutor，客户端提交的各类请求，都会被封装成 Procedure，然后提交给ProcedureExecutor中初始化运行的至少 16 个 WorkerThread中

2、提交逻辑
获取链接（初始话 Registry 和 RpcClient 两个重要的对象，分别负责和 ZK 和 Hmaster 打交道）
然后通过 MasterCallable提交 Request给 HMaster
然后通过 ClientServiceCallable提交 Request给 HRegioinServer

3、Procedure 的执行逻辑： 最终每个 Procedure 都有可能有多个不同的状态，最终是由状态机：StateMachineProcedure来协调执行。最底层其实就是执行  Procedure  的 executeFromState()
```

## 5.1.2. put 写数据流程分析

入口：HTable.put(put)

详细流程：

```
# 注意：callable = ClientServiceCallable extends RegionServerCallable
RpcRetryingCallerImpl.callWithRetries(callable, ...)

    # 第一步： regioin 定位和建立 RPC 链接
    callable.prepare(...)

        # 第一步： region 定位
        regionLocator.getRegionLocation(row)

            # 第一步：定位 meta region 的位置
            locateMeta(tableName, useCache, replicaId);

            # 第二步：定位 user region 的位置
            locateRegionInMeta(tableName, row, useCache, retry, replicaId);

        # 第二步： 建立 RPC 链接
        setStubByServiceName(this.location.getServerName());

    # 第二步： 执行 Mutation 请求处理
    callable.call(...)
        rpcCall()
            doMutate(request);
                RSRpcServices.mutate(getRpcController(), request);

                    # 第一步： 在该 RegioniServer 内部定位 Region
                    region = getRegion(request.getRegion());

                    # 第二步： 执行 Mutate 动作：Put
                    region.put(put);
                        checkReadOnly();
                        checkResources();
                            # 重要的动作：检查资源情况，执行 Flush 动作
                            requestFlush();
                        doBatchMutate(put);
                            checkReadOnly();
                            checkResources();
                            # 执行 Put 动作
                            doMiniBatchMutate(batchOp);
                                # 1、获取行锁

batchOp.lockRowsAndBuildMiniBatch(acquiredRowLocks);
                                # 2、更新时间戳
                                batchOp.prepareMiniBatchOperations(..., now,
...);
                                # 3、构造 WALEdits 对象
                                batchOp.buildWALEdits(miniBatchOp);
                                # 4、记录操作日志
                                writeEntry = doWALAppend(walEdit,....)
                                # 5、写入数据到 Memstore
                                batchOp.writeMiniBatchOperationsToMemStore(...);
                                # 6、完成写入操作
                                batchOp.completeMiniBatchOperations(...)
                            # 检查是否有必要执行 Flush
```

```
                                requestFlushIfNeeded
```

核心流程：

```
代码入口：
table.put(new Put())

大致流程：
1、首先客户端会请请求zookeeper拿到meta表的位置信息，这个meta表的region到底在那个
regionserver1里面
2、发请求请求这个regionserver1扫描这个meta表的数据，确定我要插入的数据rowkey到底在那个用户表
的region里面。并且还拿到这个region在那个regioinserver2的信息
3、发送请求，请求regionserver2扫描 当前用户表的regioin, 执行插入
    1、先记录日志
    2、写入数据到 memstore
        写到  ConcurrentSkipListMap delegatee
    3、判断是否需要进行flush
        1、再次判断是否需要进行 compact
        2、判断是否需要进行 split
```

```
batchMutate(BatchOperation<?> batchOp)
作用：完成一批次 Mutation 操作
参数：Mutation[] ===> BatchOperation
```

三大核心和步骤：

```
1、checkResources();
2、doMiniBatchMutate(batchOp);
3、requestFlushIfNeeded();
```

## 5.1.3. HBase Region 定位

入口：RegionServerCallable.prepare()

```
connection.locateRegion(){
    locateMeta(tableName, useCache, replicaId);
    locateRegionInMeta(tableName, row, useCache, retry, replicaId);
}
```

meta 表的 region 的定位的入口方法：

```
locateMeta(tableName, useCache, replicaId);
```

这个核心逻辑是怎样的呢

```
1、因为客户端会有缓存各个表的region的位置的信息

2、当客户端第一次发起put/get请求的时候，是没有缓存信息的，所以，先调用 locateInMeta()扫描
meta表获取用户表中的 rowkey对应的regioin的位置信息。这样的话，就必须要知道 meta 的region
的位置

3、在通过 locateMeta 方法来定位 meta 表的 region 的位置
具体的方法是： 因为 meta 的 region 的个数相对来说，比较少，然后每个region 的位置，都通过一个
zookeeper 的 znode 来存储。所以 locateMeta 方法，其实获取的就是这个 meta表的所有的
region 的位置信息，然后缓存在 客户端的缓存中

4、如果不是第一次发起请求，则可以优先从缓存中，获取对应的rowkey的region的位置。如果缓存失效，
再走一遍 第二步 和 第三步
```

### 5.1.4. get / scan 读数据流程分析

入口：HTable.get(get)

大致流程：

```
1、首先客户端会请请求zookeepre拿到meta表的位置信息，这个meta表的region到底在那个
regionserver1里面
2、发请求请求这个regionserver1扫描这个meta表的数据，确定我要插入的数据rowkey到底在那个用户表
的region里面。并且还拿到这个region在那个regioinserver2的信息
3、发送请求，请求regionserver2
    1、首先去blockcache ,进行查询，读缓存
    2、先去布隆过滤器中进行判断
        1、如果判断这个rowkey不存在，则不需要扫描 HFile
        2、如果判断这个rowkey存在，则需要扫描HFile
```

## 5.2. HBase 内部核心机制详解

### 5.2.1. flush 流程分析

入口：

```
1、执行put之前，检查资源：HRegion.requestFlush();
2、执行put之后，检查资源：reqeustFlush()
```

详细过程：

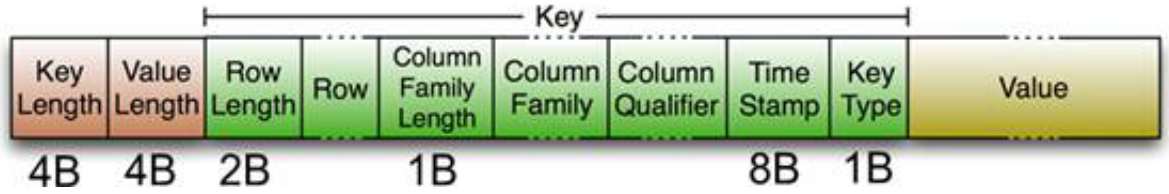
```
reqeustFlush() | requestFlushIfNeeded();
    requestFlush0(FlushLifeCycleTracker.DUMMY);
        MemstoreFlusher.requestFlush()
            fqe = new FlushRegionEntry(r, forceFlushAllStores, tracker);
            flushQueue.add(fqe);
```

把 FlushRegionEntry 加入到 flushQueue 队列中，该队列中的任务由 MemstoreFlusher 中的 FlushHander 来执行真正的 Handler

```
FlushHandler.run()
    fqe = flushQueue.poll(threadWakeFrequency, TimeUnit.MILLISECONDS);
    flushRegion(fre)
```

```
flushRegion(fqe)
    region.flushcache()
        internalFlushcache(....)
            internalPrepareFlushCache()
            internalFlushCacheAndCommit()
```

最终 HFile 文件的格式：




HBase 在逻辑上，是一个四维表稀疏表。但是物理上，其实是一个 Cell 的序列化文件

HFile 里面除了包含必要的索引信息以外，还包含很多的 Cell 对象

## 5.2.2. split 流程分析

入口：

```
compactSplitThread.requestSplit(region)
    CompactSplit.requestSplit()
```

要点：要执行 Split 动作的 RegionServer 发送请求给 Master，然后Master 来执行 Split， 最终通过状态机来执行 SplitTableRegionProcedure 的 executeFromState 方法。

详细流程：见源码！

## 5.2.3. compact 流程分析

入口：compact的入口，总共有三个

```
# 当flush完毕之后，因为增加了一个StoreFile，所以有可能要触发Compaction动作
flushResult.isCompactionNeeded();

# 当开始FLush之前，判断StoreFile文件超过指定标准的时候，则执行compaction
compactSplitThread.requestSystemCompaction(region,
Thread.currentThread().getName());

# 如果没有触发，还有定时任务执行关于 StoreFile 的定时检查
CompactionChecker定时任务
```

详细流程：见源码

其中关于：Compaction 的初始化是在： HRegioinServer 的 initializeThreads() 方法中完成的。

```
HRegionServer.start()
HRegionServer.run()
    handleReportForDutyResponse(w);
        startServices();
            initializeThreads();
                this.cacheFlusher = new MemStoreFlusher(conf, this);
                this.compactSplitThread = new CompactSplit(this);
                new CompactionChecker(this, this.ccFrequency, this);
```

# 6. 本次课程总结

今天这次课程主要讲解 HBase-2.x 的 DDL DML 语句的执行流程

```
1、CreateTable动作
2、put动作
    region定位
    flush动作
    compact动作
    split动作
3、get动作还没讲
```

这是 HBase 的核心工作机制。

# 7. 本次课程作业

当时学完 Zookeeper 的时候，给大家布置了作业，让各位实现一个类似于 ZooKeeper 的数据模型系统，具备一些基本的功能，比如：

```
1、通过树形方式来组织数据，具备基本的节点增删改查的功能
2、该系统具备冷启动恢复数据状态的功能。
```

现在来实现一个 HBase 存储系统！两点要求：

```
1、设计一个系统存储key_value类型数据，该系统为四维表模型，让系统具备根据一个条件，两个条件，三个条件，四个条件查找value的能力，如果不是根据四个条件来查找，返回单个value，如果不是根据四个条件来查询，则返回value的集合，最好不要只是value的集合，最好是一个map。
2、该系统可以冷启动恢复数据状态
```